

Name 1:

Name 2:

---

# TCP/IP NETWORKING

## LAB EXERCISES (TP) 5

### CONGESTION CONTROL; TCP, UDP

---

Friday, November 23, 2018

#### Abstract

In this lab session, you will explore in a virtual environment the effect of the congestion control mechanism of TCP and compare with a situation without congestion control. You will see what types of fairness are achieved by this congestion control mechanism. You will observe that a congestion control mechanism is also essential to avoid congestion collapse<sup>1</sup>.

## 1 IMPORTANT NOTES FOR THIS LAB EXERCISE

During this lab, **you will use the virtual machine that you setup in the first three labs** and you will study various aspects of congestion control in several network topologies constructed by Python scripts in Mininet.

**Important Note:** For this lab, your virtual machine should use the virtual HD image *Lubuntu16.vdi* (follow the procedure of Lab 1). *Lubuntu16.vdi* has a Linux version that supports DCTCP, which will be studied in the research exercise.

### 1.1 REPORT

This document will be your report (one per group). Type the answers directly in the PDF document. Use Adobe Reader XI, as it supports saving forms. When you finish, upload the report on Moodle. Do not forget to write your names on the first page of the report. **The deadline is Wednesday, December 5th, 23:55pm.**

For the research exercise, a separate report in pdf should be submitted. Please be short and precise in your answers.

### 1.2 PRELIMINARY INFORMATION

1. On the virtual machine (VM), you also need to download an archive that contains the programs for this lab. Download the file `lab5.zip` from Moodle (<http://moodle.epfl.ch/course/>

---

<sup>1</sup>In fact, this is the primary role of a congestion control algorithm. See *Congestion Avoidance and Control*. Van Jacobson and Michael J. Karels. ACM SIGCOMM 1988

[view.php?id=523](#)), copy it and uncompress it in the shared folder. The folder `lab5` contains three folders. The folder `lab5/scripts/` contains the python scripts that will be used to build the topologies of this lab for the experiments that will run in Mininet. The folders `lab5/tcp/` and `lab5/udp/` contain `tcp` and `udp` (correspondingly) clients and servers that we will use to measure the goodput of each flow.

2. When needed, make the programs executable by going in the directory `lab5` (by typing `cd lab5`) and typing:

```
chmod +x tcp/tcpclient tcp/tcpserver udp/udpclient udp/udpserver
```

**Note:** All folders contain binary programs as well as the source code. Though, the binaries should work in your VM and you should not need to recompile the source codes. If you want to (or need to) recompile them, you will probably need to install a few packages, including `gcc`, `make`, and `linux-module-headers`. Then, each program can be compiled by typing `make` in its own directory.

3. Last, we need to check and/or modify the congestion control mechanism. On a Linux machine, you can test which congestion control mechanism is used by typing in a terminal:

```
cat /proc/sys/net/ipv4/tcp_congestion_control
```

In this lab, we will force TCP to use the CUBIC congestion control algorithm. If the congestion control algorithm is not Cubic, you **should** change it to Cubic *until* the next reboot by typing in a terminal:

```
echo cubic >/proc/sys/net/ipv4/tcp_congestion_control
```

Similarly you can set the RENO or the DCTCP congestion control algorithm, i.e.,

```
echo reno >/proc/sys/net/ipv4/tcp_congestion_control
```

```
echo dctcp >/proc/sys/net/ipv4/tcp_congestion_control
```

4. **Important Note:** In Mininet, between two directly connected hosts/routers there is always a 'transparent' switch, which may not be shown in the figures but it is already added in the code.

## 2 TCP VS UDP FLOWS

The topology that is used in this section is shown in Fig. 1. You will construct this topology in Mininet by completing the script `lab5/scripts/lab51_network.py`.

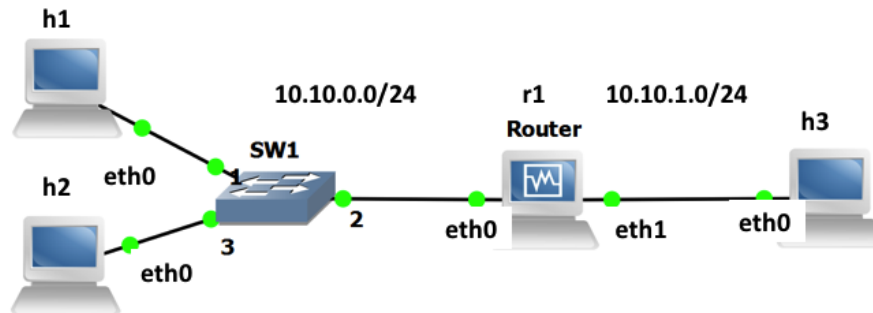


Figure 1: Initial configuration with 3 PCs and one router.

Complete the file `lab51_network.py` (in the two designated spaces) so as to configure the IP addresses and the routing tables according to the following addressing scheme:

- The subnet of hosts h1, h2 and the router (r1) is 10.10.0.0/24. The addresses of h1, h2 and of the router are respectively 10.10.0.1, 10.10.0.2 and 10.10.0.10.
- The subnet of h3 and of the router is 10.10.1.0/24. The addresses of h3 and of the router are respectively 10.10.1.3 and 10.10.1.10.

Open a terminal in your VM and run the script `lab51_network.py`. Test your configuration after running the topology with the `pingall` command.

### 2.1 TESTING THE CONNECTIVITY WITH BASIC UDP AND TCP CLIENTS/SERVERS

The directory `lab5/udp` contains two programs: `udpserver` and `udpclient`. Their usage is:

```
# ./udpserver PORT
# ./udpclient IP_SERVER PORT RATE
```

For the server, `PORT` is the port number on which the server listens. For the client `IP_SERVER` and `PORT` are the IP address and port of the machine to which the packets are sent and `RATE` is the rate at which the client sends data (in kilobits per seconds). The client sends packets of size 125 bytes if the rate is lower than 50kbps and of size 1000 bytes otherwise.

The output of the UDP client has the following format:

```
4.0s - sent: 503 pkts, 1000.0 kbits/s
5.0s - sent: 629 pkts, 1000.6 kbits/s
```

5.0 is the number of seconds since the launching time of the client, 629 is the total number of packets sent by the client and 1000.6 is the sending rate during the last second (in kilobits per second).

The output of the UDP server has the following format:

```
169.5s - received: 723/ sent: 741 pkts (loss 2.429%), 959.6 kbit/s
170.5s - received: 843/ sent: 867 pkts (loss 2.768%), 957.7 kbit/s
```

The values of the second line are explained as: 170.5 is the time since the launching of the server, 843 and 867 are the total number of packets sent by the client and received by the server, 2.768% is the percentage of packets that were lost and 957.7 is the rate at which packets were received during the last second. The latter value is defined as the goodput at the last second (see also the remarks in Section 2.1.1).

Start a UDP server on host h3 that listens on port 10000.

**Remark:** If you run the experiment multiple times the results may vary since Mininet is a network emulator. Thus, it is highly recommended that you run each experiment multiple times (e.g., 5) and provide the averaged values as the answer.



**QQ1/** Launch a UDP client on host h1 that sends data to this server at rate 100 kbps. What are the loss percentage and the goodput that you observe on h3?

[A1]

The directory `lab5/tcp` contains two programs: `tcpserver` and `tcpclient`. Their usage is similar to `udpserver` and `udpclient`, except that we do not specify a rate to the client: the client has an unlimited amount of data to send and uses TCP congestion control algorithm to control at which rate it sends the data to the server.

```
# ./tcpserver PORT
# ./tcpclient IP_SERVER PORT
```

The output of a TCP client looks like this:

```
6.3: 854.0 kbps avg ( 944.5[inst], 926.5[mov.avg]) cwnd 9 rtt 83.9ms
7.3: 862.4 kbps avg ( 914.6[inst], 925.3[mov.avg]) cwnd 9 rtt 86.8ms
```

The values of the second line are explained as: 7.3 is the time, 862.4 is the average rate of the client: the total amount of data that was successfully transferred by the client divided by the total time (this is defined as goodput - average value - for the TCP). 914.6 is the instantaneous rate (approximately over the last second) and 925.3 is a moving average of this value. The value 9 is the congestion window of the TCP connection and 86.8 is the RTT measured by the TCP congestion control algorithm.

Start a TCP server on host h3 that listens on port 10001.



**QQ2/** Launch a TCP client on host h1 that sends data to this server. What is the goodput of the connection?

[A2]

### 2.1.1 REMARKS ON THE PROGRAMS `UDPCLIENT`, `TCPCLIENT`, `UDPSEVER` AND `TCPSEVER`

The directories `lab5/tcp/` and `lab5/udp/` contain the executable and the source code of the programs.



• **For each UDP flow, you need one UDP client and one UDP server.** Explanation: each packet sent by a client contains its sequence number (the first packet contains the label “1”, the second “2”,...) and a lot of “0” to reach a size of 1000 bytes or 125 bytes. The loss percentage printed by the server is one minus the ratio between the number of packets received divided by the largest sequence number received. Because of this implementation, the loss percentage printed by the server is wrong if two clients talk to the same server.

- **For TCP, one server can handle multiple clients.** The server creates one thread per accepted connection.
- **Before each experiment, kill all clients (TCP and UDP)** (you can do that by pressing “Control-C” in the terminal of the client.) This will reset the average values printed by the clients. In theory, you can keep the server running but killing them and relaunching them will not harm.
- **The printed rates correspond to application data.** They count the amount of data that was transferred by the TCP/UDP client to the TCP/UDP server. They do not take into account headers.
- **For all experiments, you have to wait until the printed values stabilize.** This is particularly important for TCP. The rate at which TCP sends packets depends on the losses that occur at random. Thus, to obtain deterministic values, you should wait for the average rate to be stable (5 minutes is probably OK for most scenarios). We also encourage you to run the experiments multiple times and provide average values.
- **Goodput.** The goodput of a flow is the rate of *application data* (i.e., useful data) that is successfully transmitted. For the theoretical questions, you should take into account that the packets also contain header except from the application data.
- **Units.** In all your answers, indicate in which unit your result is expressed (Mbps, kbps, %, ...).
- **Queueing delay.** It is defined as the time (in the appropriate unit) that a packet waits stored in the queue of a router until it is forwarded.

### 2.2 ARTIFICIAL LIMITATION OF THE BANDWIDTH OF THE ROUTER

In order to produce experiments where the performance is limited by the network capacities, we will further limit the bandwidth of some interfaces in the topology. To do so, in this section we use the possibilities offered by the `TCLink` class in Mininet.

In the script `lab51_network.py`, which you completed and run in the previous section, you can find the command

```
link_r1sw2.intf1.config( bw=10, enable_red=True, enable_ecn=True )
```

The part `bw = 10` of this command configures the bandwidth of the interface `r1 – eth1` of the router to 10 Mbps. Modify the command so that the bandwidth of the interface `r1 – eth1` of the router is bounded to 4 Mbps.

**Note:** You should exit Mininet, clean up the topology of the previous section and run the script `lab51_network.py` with the new bandwidth configuration. The script should involve the additions made in Section 2.1 so that the topology in Fig. 1 is connected and all hosts communicate.

**Remark:** The limit of 4 Mbps is both for the packets that come in and go out of the interface `r1 – eth1` of the router, since the `TCLink` class creates symmetric TC link interfaces. It is possible to perform bandwidth

limitations with the linux tools by using the *class based queueing* (CBQ) scheduler. These tools allow for non-symmetric bandwidth limitations. The rest of the remark is for the interested readers only otherwise continue with Section 2.2.1.

As an example of a CBQ configuration, we postulate and explain the following commands:

```
# tc qdisc add dev r1-eth1 root handle 1: cbq avpkt 1000 bandwidth
60mbit
# tc class add dev r1-eth1 parent 1: classid 1:1 cbq rate 2mbit allot
1500 prio 5 bounded isolated
# tc filter add dev r1-eth1 protocol ip parent 1: prio 16 u32 match ip
src 0/0 flowid 1:1
```

The first line installs a class based queue on r1-eth1, and tells the kernel that for calculations, it can be assumed to be a 60 Mbps interface. The exact value is less important as we will be limiting the rate below 60 Mbps.

The second line creates a class, with ID 1 : 1, which has 2 Mbit per second limit on the bandwidth allocated to it (and some other default parameters).

The third line tells which traffic should go to the shaped class. In particular, it says that all packets going out of r1-eth1 will be stored in the class 1 : 1.

For more information see: <http://linux.die.net/man/8/tc-cbq> or the tutorial in <http://lartc.org/howto/lartc.qdisc.classful.html>.

In case you need to change the configuration of the CBQ, you first need to delete the previous configuration. Do so by using the command `tc qdisc del dev eth1 root`. The limit of 2 Mbps is only for the packets that go out of the interface r1-eth1. Therefore, the packets coming towards the router do not interfere with the 2 Mbps limit.

### 2.2.1 UDP TEST

When the RATE at which the UDP client sends data is greater than 50 kbps, the client sends packets that contain 1000 bytes of data each.



**QQQ3/** What is the size (in bytes) of the Ethernet frames that you expect to be used for sending the data of the UDP client? Verify with wireshark at the server side.

[A3]

**QQQ4/** The router is the bottleneck and has a limit of 4 Mbps. What is the maximum theoretical aggregate application data throughput (i.e., goodput) that can be achieved? Explain how you compute.

[A4]

Start a UDP server on host h3 that listens on port 10000.



QQQ5/ Start a UDP client on host h1 that sends data at rate 100 kbps. What are the loss percentage and the goodput observed on h3?

[A5]

QQQ6/ Repeat the operation with 4 Mbps, 10 Mbps and 20 Mbps. What are the goodputs and loss percentages? What do you observe (explain)? Compare to the theoretical goodput computed above.

[A6]

### 2.2.2 TCP TEST



QQQ7/ What is the size (in bytes) of Ethernet frames that you expect to be used for sending the data by the TCP connection? Verify with wireshark at the server side.

[A7]

QQQ8/ What is then the maximum theoretical aggregate application data throughput (i.e., goodput) that can be achieved? Explain how you compute.

[A8]

Start a TCP server on host h3 that listens on port 10001.



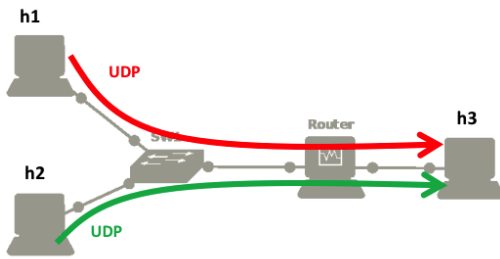
QQQ9/ Launch a TCP client on host h1 that sends data to this server. What is the goodput of the connection? Compare to the theoretical goodput computed above.

[A9]

## 2.3 COMPETING UDP FLOWS

We now want to explore what happens when two UDP flows are competing for the same bottleneck. **The router should have a capacity of 4 Mbps and is the bottleneck.** We consider the following scenarios:

- Host h1 is streaming real-time data (*e.g.*, coming from a Phasor Measurement Unit (PMU)) at rate 2 Mbps to host h3 using UDP.
- Host h2 is streaming a video to host h3 using UDP. Depending on the quality, h2 sends at rate 1 Mbps, 2 Mbps or 5 Mbps.



Scenario	h1 (UDP)	h2 (UDP)
A1	2 Mbps	1 Mbps
A2	2 Mbps	2 Mbps
A3	2 Mbps	5 Mbps

Before doing measurement, we want to predict the amount of data that will be sent and received in the three scenarios (denoted A1, A2 and A3).

**QQ10/** Based on a theoretical analysis, what should be the goodputs and loss percentages of hosts h1 and h2 in the scenarios A1, A2 and A3. Explain your method below and fill in the table.

[A10]

	Goodput h1	Loss percentage h1	Goodput h2	Loss percentage h2
Q10a/ (A1)	[10a(i)]	[10a(ii)]	[10a(iii)]	[10a(iv)]
Q10b/ (A2)	[10b(i)]	[10b(ii)]	[10b(iii)]	[10b(iv)]
Q10c/ (A3)	[10c(i)]	[10c(ii)]	[10c(iii)]	[10c(iv)]

We now want to verify our analysis via emulation. For each scenario, start two UDP servers on h3 that listen on ports 10001 and 10002. Use the command `xterm &` at h3's terminal to open a new terminal for h3. Then, run a UDP client on h1 that sends data to h3 at 2 Mbps and a UDP client on h2 that sends data to h3 at rate 1, 2 or 5 Mbps.

**QQ11/** What are the measured goodputs and loss percentages in scenarios A1, A2 and A3?

[A11]

	Goodput h1	Loss percentage h1	Goodput h2	Loss percentage h2
Q11a/ (A1)	[11a(i)]	[11a(ii)]	[11a(iii)]	[11a(iv)]
Q11b/ (A2)	[11b(i)]	[11b(ii)]	[11b(iii)]	[11b(iv)]
Q11c/ (A3)	[11c(i)]	[11c(ii)]	[11c(iii)]	[11c(iv)]

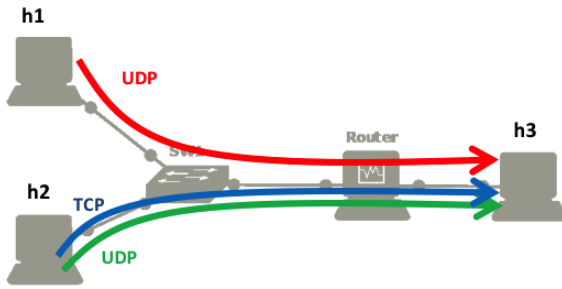
**QQ12/** Do you see a difference between your theoretical analysis and the emulation? If yes, comment on the difference, and try to explain the possible sources.

[A12]

## 2.4 TCP FLOWS COMPETING WITH UDP FLOWS

We consider a similar scenario as in Section 2.3. Host h1 streams PMU data at rate 2 Mbps to host h3 and h2 streams video data to h3 at rate 1 Mbps, 2 Mbps or 5 Mbps. In addition to this traffic, host h2 is also using a TCP connection to send a software update to h3.





Scenario	h1 (UDP)	h2 (UDP)
B1	2 Mbps	1 Mbps
B2	2 Mbps	2 Mbps
B3	2 Mbps	5 Mbps

**QQ13/** Based on a theoretical analysis, what should be the goodputs of the UDP flow of h1, the TCP flow of h2 and the UDP flow of h2 in the scenarios B1, B2 and B3 (explain below and fill in the table)?

[A13]

	UDP flow of h1	UDP flow of h2	TCP flow (h2)
Q13a/ (B1)	[13a(i)]	[13a(ii)]	[13a(iii)]
Q13b/ (B2)	[13b(i)]	[13b(ii)]	[13b(iii)]
Q13c/ (B3)	[13c(i)]	[13c(ii)]	[13c(iii)]

We now want to verify our analysis via an emulation. As in the previous case, the bandwidth is limited to 4Mbps (outgoing packets of r1-eth1) and we start two UDP servers on h3. Start also a TCP server on h3. Then, start the corresponding clients in hosts h1, h2. Use the command `xterm &` at h2's terminal to open a new terminal for h2.

**QQ14/** What are the measured goodputs of the three flows in scenarios B1, B2 and B3?

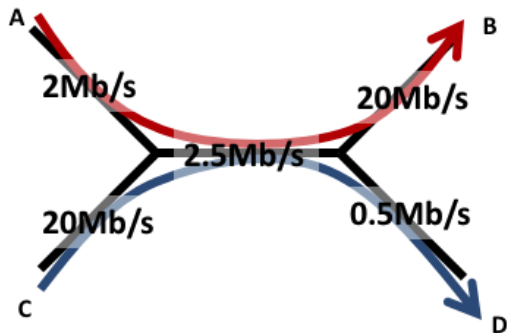
[A14]

	UDP flow of h1	UDP flow of h2	TCP flow (h2)
Q14a/ (B1)	[14a(i)]	[14a(ii)]	[14a(iii)]
Q14b/ (B2)	[14b(i)]	[14b(ii)]	[14b(iii)]
Q14c/ (B3)	[14c(i)]	[14c(ii)]	[14c(iii)]

**QQ15/** Do you see a difference between your theoretical analysis and the emulation? If yes, comment on the difference, and try to explain the possible sources.

[A15]

### 3 THE IMPORTANCE OF CONGESTION CONTROL




In this section, we will explore why having a congestion control mechanism is necessary. The system that we want to emulate is composed of five links depicted on the left. The capacities of the links range from 0.5Mbps to 20Mbps. There are two flows in this network:

- one flow that goes from A to B (in red),
- one flow that goes from C to D (in blue).

We will show evidence of a phenomenon called *congestion collapse*: the more aggressive C is, the smaller the total goodput will be.


#### 3.1 THEORETICAL ANALYSIS

We first assume that there is no congestion control. The two senders, A and C, send data using UDP.

 **QQQ16/** If both sender A and sender C try to send data at maximum speed (i.e. 2Mbps and 20Mbps), what are the goodputs received by B and D? What are the loss percentages of these two links?

[A16]

We now assume that sender A and sender C use a congestion control mechanism.

 What is the rate at which A and C will send data if we use **QQQ17/** a max-min fair allocation?

[A17]

**QQQ18/** a proportionally fair allocation?

[A18]

#### 3.2 EXPERIMENTAL SETTING

We now want to verify these results in the virtual environment.

The script `lab53_network.py` creates a new topology according to Figure 2.

The addressing scheme is as follows:

- The addresses of h1, h2, h3 and h4 end with 1, 2, 3 and 4.
- The addresses of the routers 1, 2 end with 10, 20.

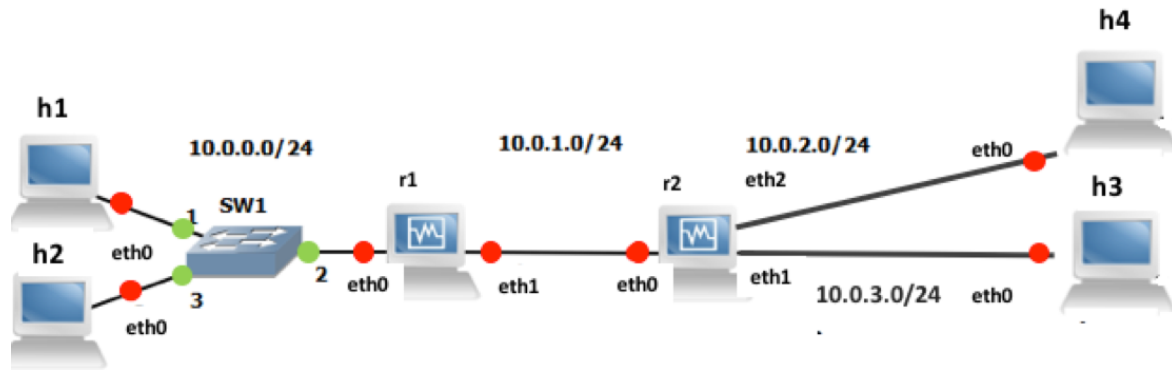



Figure 2: Setting for studying the congestion collapse.

The bandwidth limits of the links are already set at the script. ECN and RED are also enabled at the routers. More information on the ECN can be found in the research exercise in Section 6.

After running the script you can verify that the configuration works with the pingall command.


### 3.2.1 UDP

Launch a udp server on host h3 and another one on host h4 that listen on ports 10001 and 10002 correspondingly. Launch two UDP clients, one on host h1 sending to host h4 and one on host h2 sending to host h3 with rates 2Mbps (h1) and 20Mbps (h2) correspondingly.

 **QQ19/** What are the goodputs of the two flows? What are the loss percentages?

[A19]


Now, launch two UDP clients, one on host h1 and one on host h2, that send data according to the max-min fair allocation that you computed before.

 **QQ20/** What are the goodputs of the two flows? What are the loss percentages?

[A20]

### 3.2.2 TCP

Repeat the same process using TCP connections instead of UDP.

 **QQ21/** What are the goodputs of the two connections? Did you expect the observed rates?

[A21]



QQQ22/ Can you conclude on what are the advantages of having a congestion control mechanism?

[A22]

## 4 TCP: FAIRNESS AND INFLUENCE OF RTT

The congestion control algorithm of TCP guarantees that the network resources are shared among the different connections. In this part, we will explore how the CUBIC algorithm shares the bandwidth when one or multiple bottlenecks are present in the network. In particular, we will investigate two characteristics: (i) CUBIC provides a fairness *per flow* and (ii) when the RTT or delay-bandwidth product values are low, Cubic is sensitive to delay and when the RTT or delay-bandwidth product values are high Cubic is less sensitive to delay. Note that for low RTT values CUBIC performs similarly to RENO, while this is not the case for higher RTT values as it is explained in the lecture notes. Also, we will perform comparisons between TCP CUBIC and TCP RENO.



For Sections 4.1 and 4.2, we will reuse the setting of Figure 1, which is created by the script `lab51_network.py`. The bandwidth of the router (r1-eth1) should be limited to **4 Mbps** (use the same configuration as in Section 2.2). Test the connectivity of your topology with the `pingall` command.



In this part in particular, it is important to wait until the printed goodputs stabilize. To speedup the convergence, it is **very** recommended to close all the unnecessary programs on your computer. Especially, you should close the programs that may perform things on background (such as web-browsers, Dropbox synchronization, other virtual machines, etc). In any case, you should wait around 2-5 minutes to see the stable results.

For the whole section, in order to enable better convergence that will be less impacted by queueing delays that affect the RTT values, **ECN and RED are already enabled in the script `lab51_network.py`**.


### 4.1 ADDING DELAY TO AN INTERFACE

To obtain more realistic and more reproducible experiments, we will add delay in the network. To do so, we will use the module `netem` of the software *traffic control* that exists in Linux. We can use the command `tc` to add a rule in order to delay packets on an interface (see <http://www.linuxfoundation.org/collaborate/workgroups/networking/netem> for more information about `tc` and `netem`).

For example, the following command adds 300 ms of delay to all packets going out of the interface `eth0` (one direction only, not applied to the packets coming in!!!):

```
# tc qdisc add dev eth0 root netem delay 300ms
```

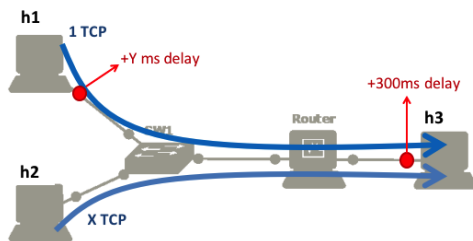
This rule can be changed to e.g., 400ms by typing `tc qdisc change dev eth0 root netem delay 400ms` or deleted by typing `tc qdisc del dev eth0 root`.

 **QQ23/** Add 300 ms of delay to the interface h3-eth0 of h3. Ping host h1 from host h3. What is the observed RTT?

[A23]

**Remark:** if you flush the ARP table (for example, by using the commands `ifconfig h3-eth0 down` && `ifconfig h3-eth0 up`) and you reconfigure `netem` to add 300 ms, the RTT of the first packet should be larger than the RTT of the second packet, because of the ARP request.


## 4.2 FAIRNESS BETWEEN TCP CONNECTIONS AND DELAY



Scenario	X (# conn. on h2)	Delay Y
D1	3	0 ms
D2	1	400 ms
D3	1	800 ms


TCP provides a fair sharing of the bandwidth at the flow level. Therefore, a machine that opens several TCP connections will obtain more bandwidth. To verify that, we will use the scenario D1:

- There is an additional delay of 300 ms on the interface h3-eth0 of host h3 but none on hosts h1 or h2.
- Host h1 opens one TCP connection to h3 and host h2 opens three TCP connections to h3.

 **QQ24/** Using a theoretical analysis, what is the total goodput that host h1 and host h2 will get in scenario D1?

[A24]

Start a TCP server on host h3. Run the 3 TCP clients on host h2 and one TCP client on h1. Wait until the rates stabilize.

 **QQ25/** What are the aggregate goodputs obtained by h1 and by h2 in scenario D1? Does this correspond to your theoretical analysis and if there is a difference, can you explain why?

[A25]

**QQQ26/** Can you tell if there is any queuing delay?

[A26]

We now explore scenarios D2 and D3, where hosts h1 and h2 both open 1 TCP connection to host h3. Assume that the RTT is 300ms for the connections coming from h2 and  $(300 + Y)$ ms for the connections coming from h1.



**QQQ27/** For TCP RENO, in theory, what is the goodput that h1 and h2 will get as a function of  $Y$ ?

[A27]

Numerically, what are these values when:

	Total goodput for h1	Total goodput for h2
Q27a/ (D2) $Y = 400$ ms	[27a(i)]	[27a(ii)]
Q27b/ (D3) $Y = 800$ ms	[27b(i)]	[27b(ii)]



**QQQ28/** Run the simulation corresponding to scenarios D2 and D3 for TCP CUBIC. What are the measured goodputs obtained by h1 and by h2?

[A28]

	Total goodput for h1	Total goodput for h2
Q28a/ (D2)	[28a(i)]	[28a(ii)]
Q28b/ (D3)	[28b(i)]	[28b(ii)]

**QQQ29/** Now, run the simulation corresponding to scenarios D2 and D3 for TCP RENO. For TCP RENO what are the measured goodputs obtained by h1 and by h2.

[A29]

	Total goodput for h1	Total goodput for h2
Q29a/ (D2)	[29a(i)]	[29a(ii)]
Q29b/ (D3)	[29b(i)]	[29b(ii)]

**QQQ30/** Do the results for TCP RENO correspond to your theoretical analysis? If there is a difference, can you explain why?

[A30]

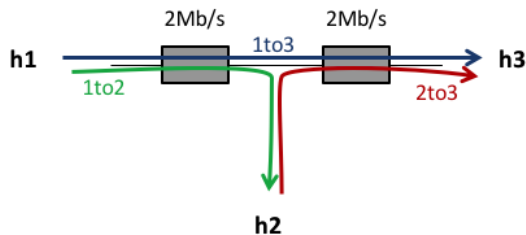
**QQQ31/** Can you explain the difference observed between the results of TCP RENO and TCP CUBIC? Can you explain at which regime TCP CUBIC works, namely, (i) in the regime where the RTT or product delay-bandwidth values are considered low or (ii) in the regime where the RTT or product delay-bandwidth

values are considered high? Explain.

[A31]

At this moment, change back your TCP algorithm to TCP CUBIC.

### 4.3 FAIRNESS OF TCP CONNECTIONS TRAVERSING MULTIPLE BOTTLENECKS



In this part, your goal is to study how the available bandwidth is shared when (i) one TCP connection traverses two bottlenecks (ii) each of the bottlenecks is also traversed by another TCP connection, as shown in the figure.

The notion of fairness is difficult. A rate allocation is always a trade-off between maximizing the total rates sent by the connection or trying to equalize the rates of all users. For example, in this scenario, the flow *1to3* uses twice more resources than the flows *1to2* and *2to3*. Thus, the bigger the traffic *1to3* is, the lower the aggregate goodput can be.

#### 4.3.1 THEORETICAL ANALYSIS

We first perform a theoretical analysis to compute two *fair* allocations corresponding to this network.



Using a theoretical analysis:

QQQ32/ What is the max-min fair allocation that corresponds to this network (explain)?

[A32]

QQQ33/ What is the proportionally fair allocation (explain)?

[A33]

#### 4.3.2 EXPERIMENTAL SETTING

We now want to explore what is the allocation provided by TCP.

The script `lab54_network.py` constructs the topology according to Figure 3 (note: the router 2 needs to have 3 interfaces available). Complete the script `lab54_network.py` by setting the routing tables at the routers, the default gateways of the hosts and by configuring bandwidth and delay values as shown in the following figure. **ECN and RED are already enabled in the script at the routers 1 and 2 for faster and better convergence.**

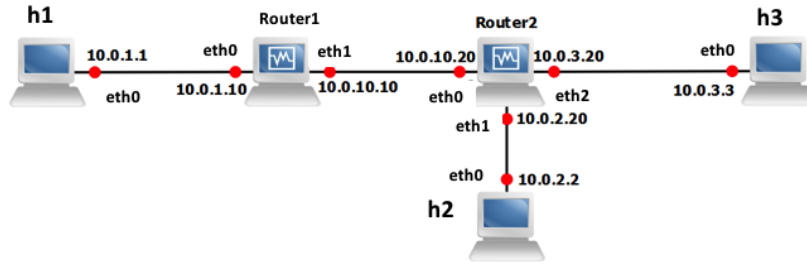
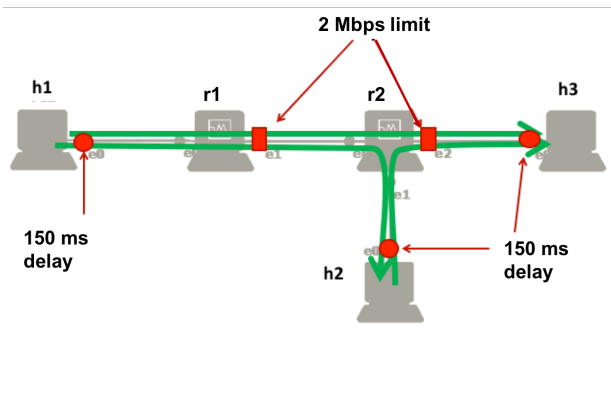


Figure 3: Fairness of TCP connections traversing multiple bottlenecks: wiring and addressing scheme.



The bandwidth values of the interfaces can be configured as in Section 2.2 of this lab: the bandwidth of the interfaces eth1 of Router1 and eth2 of Router2 are limited to 2 Mbps by appropriately setting the bw feature.

As in Section 4.1 of this lab, use `tc` to add 150 ms of delay to the outgoing packets of the interfaces h1-eth0, h2-eth0 and h3-eth0.

**QQ34/** Assume that there is no queuing delay. In theory, what should be the RTT of the connections *1to2*, *2to3* and *1to3*?

[A34]

Now, start a TCP server on h2 and on h3. On host h1, open one TCP connection to h2 and one TCP connection to h3. On host h2, open one TCP connection to h3. Wait until the rates stabilize.

**QQ35/** What is the measured goodput of each one of the three connections?

[A35]

[A35.a] *1to2*

[A35.b] *1to3*

[A35.c] *2to3*

**QQ36/** Does this corresponds to your theoretical analysis?

[A36]

**QQ37/** What is approximately the average RTT of all three connections? Can you estimate approximately



the queuing delay on Router1 and Router2?

[A37]

## 5 RESEARCH EXERCISE 1: COMPARISON OF TCP CUBIC WITH TCP RENO

In this research exercise, we will continue the experiment of Section 4.3.2 (after question 37) in order to further compare the behavior of TCP CUBIC with TCP RENO. Modify the delay on h2 to 1000 ms and repeat the same experiment with question 35 in Section 4.3.2.



**Q38/** For TCP CUBIC, is the goodput of  $1to3$  larger or smaller than the ones of  $1to2$  and  $2to3$  (comment the results)?

[A38]

Now, also change your TCP algorithm to RENO and repeat the experiment of question 35.



**Q39/** For TCP RENO, is the goodput of  $1to3$  larger or smaller than the ones of  $1to2$  and  $2to3$ ? Comment the results and compare with TCP CUBIC. Explain also at which regime TCP CUBIC works, i.e., are the RTT or product delay-bandwidth values considered low or high?

[A39]

## 6 RESEARCH EXERCISE 2: STUDY OF DCTCP & COMPARISONS WITH TCP CUBIC & TCP CUBIC WITH EXPLICIT CONGESTION NOTIFICATION (ECN)

(Optional, for bonus)

ECN allows end-to-end notification of network congestion without dropping packets. Conventionally, TCP/IP networks signal congestion by dropping packets. When ECN is enabled (and in addition RED is enabled), an ECN-aware router may set a mark, i.e., the Congestion Experienced (CE) bit, in the IP header instead of dropping a packet in order to signal impending congestion. The receiver of the packet echoes (by setting the ECN Echo flag) the congestion indication to the sender, which reduces its transmission rate as if it detected a dropped packet and begins fast retransmit.

In this research exercise, we will study the effect of enabling ECN on the RTT and the congestion window by comparing CUBIC without using ECN/RED and CUBIC with ECN/RED. In this research exercise, we will also compare DCTCP with CUBIC.

**Important Note:** For this research exercise, you will write your answers and provide the appropriate graphs and explanations in a separate document, which you will upload as pdf together with this pdf file.

To achieve the above goals, we will use the topology of Fig. 1 which is defined in the script `lab51_network.py`. We should modify the script in order to limit the queue length of the router 1 by appropriately setting the `max_queue_size` parameter to the value of 100 packets. Furthermore you should limit the bandwidth of the interface 1 (r1-eth1) of the router equal to *5Mbps*.

You will run the topology `lab51_network.py` (with the two modifications above) in the three scenarios that follow.

**Scenario 1:** TCP CUBIC with **disabled ECN & RED** at the interface 1 of r1.

**Scenario 2:** TCP CUBIC with **enabled ECN & RED** at the interface 1 of r1.

**Scenario 3:** DCTCP with **enabled ECN & RED** at the interface 1 of r1. Note that for DCTCP ECN and RED should be always enabled as they are used for the estimation of the probability of congestion.

**For each of the above scenarios, you will start a TCP server on h3 and two TCP clients on hosts h1 and h2 sending traffic to h3. Also, you should open wireshark or tshark in both hosts h1 and h2 for capturing the traffic.**

Note that for emulating a new scenario, you should exit Mininet, clean up any previous topology, configure the appropriate TCP version and then run the script `lab51_network.py` with the appropriate configurations as described in each scenario. You should always wait until the rates stabilize. **After the end of the emulation for each scenario you should save the wireshark captures (name.pcapng) for hosts h1 and h2. Also note down the rates at which each emulation converged for both hosts h1 and h2 and the congestion window ranges. For tshark use the command `tshark -i h1-eth0 -F pcapng -w name.pcapng`.**

Now we will use `tcptrace` and `xplot` (which are already installed for you in the virtual HD image) on the files `*.pcapng` that you saved from wireshark or tshark for each scenario in order to compare the behavior of CUBIC without using ECN/RED, CUBIC with ECN/RED and DCTCP with ECN/RED.

The command

```
tcptrace -S name.pcapng
```

constructs two files denoted as `a2b-tsg.xpl` and `b2a-tsg.xpl`, that will be used from `xplot` as:

```
xplot.org a2b_tsg.xpl
```

to construct the Time Sequence Graph from sender a to receiver b. Please check what the Time Sequence Graph shows and how to use it in [http://www.tcptrace.org/manual/node12\\_mn.html](http://www.tcptrace.org/manual/node12_mn.html). Also, you need to use one of the two files; check the ip addresses of the sender and of the receiver on the title of the figure to choose the correct one.

Similarly, the command

```
tcptrace -R name.pcapng
```

constructs two files denoted as *a2b\_rtt.xpl* and *b2a\_rtt.xpl*, that will be used from *xplot* as:

```
xplot.org a2b_rtt.xpl
```

to construct the graph of the RTT over time from sender a to receiver b. Again, you need to use one of the two files; check the ip addresses of the sender and of the receiver on the title of the figure to choose the correct one.

Please use the screenshot tool of Linux (Accessories → Screenshot) to save these graphs.

By using all the above emulation results for both hosts and the three scenarios, compare the three scenarios in terms of the values of rates achieved, the congestion window ranges, the RTT values and the number of retransmissions. Provide appropriate explanations and graphs. Zoomed versions of graphs may be needed to show retransmissions in the time-sequence graph. Write your report while expressing yourself in a short and precise way.



Under some assumptions, which almost hold for a small number of flows as in our case, the maximum queue size that DCTCP can reach is given by  $K + N$  where  $N$  is the number of flows and  $K$  is the queue size threshold after which DCTCP starts marking packets (see M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, Data center TCP (DCTCP). SIGCOMM '10, ACM). For low rates  $\leq 1Gbps$ ,  $K$  is empirically set around 20 packets. Based on this formula and the other configurations of this experiment, explain the behavior of DCTCP with respect to the retransmissions.